SISTEMI OPERATIVI

ESERCIZIO N. 1 dell'8 NOVEMBRE 2002

TESTO 2 della PROVA IN ITINERE

In una **banca**, **C conti correnti** vengono acceduti dai **dipendenti**, i quali richiedono l'accesso ai conti correnti, li elaborano e poi li rilasciano. I dipendenti accedono a 2 conti correnti alla volta, di cui uno in lettura e uno in scrittura. In generale, la banca consente la lettura concorrente di un conto da parte di più dipendenti, mentre vincola che un conto possa essere acceduto in scrittura da un solo dipendente alla volta, ed inoltre non è possibile leggere e scrivere sullo stesso conto contemporaneamente. I dipendenti sono di due tipi: **impiegati** o **dirigenti**. I secondi hanno priorità sui primi.
Si implementi una soluzione usando il costrutto monitor per modellare la **banca** e i processi per modellare i **dipendenti** e si descriva la sincronizzazione tra i processi. Nella soluzione si massimizzi l'utilizzo delle risorse. Si discuta se la soluzione proposta può presentare starvation e in caso positivo per quali processi, e si propongano modifiche e/o aggiunte per evitare starvation.

```
program Banca

const     C = ...; { numero di conti correnti }
type      conto = 1..C;
type      tipo = (impiegato, dirigente);

type dipendente = process (t: tipo; c1, c2: conto)
begin
      repeat
            s.richiedi (t, c1, c2);
            <legge da c1 e scrive su c2 >
            s.rilascia (c1, c2);
      until false
end

type banca = monitor

{ variabili del monitor }
var  sospesi: array[tipo] of integer;
      { numero di dipendenti sospesi }
      coda : array[tipo] of condition;
      { code su cui sospendere i dipendenti }
      lettori : array[conto] of integer;
      { numero di lettori per conto corrente }
      scrittore : array[conto] of boolean;
      { dice se i conti correnti sono acceduti in scrittura }

procedure entry richiedi (t: tipo; c1, c2: conto)
begin
      while (scrittore[c1] or { se c'è uno scrittore su c1}
        lettori[c2] > 0 or { o un lettore su c2}
        scrittore[c2] or { o uno scrittore su c2}
        (t = impiegato and coda[dirigente].queue) { o c'è un
dirigente in coda }
      begin
            sospesi[t] ++;
```

```
            coda[t].wait;
            sospesi[t] --;
        end

        { acquisisce la risorsa }
        lettori[c1] ++;
        scrittore[c2] := true ;
end

procedure entry rilascia (c1, c2: conto)
var s, i: integer;
begin
        { rilascia la risorsa }
        lettori[c1] --;
        scrittore[c2] := false ;

        { risveglia prima i dipendenti dirigenti }
        s := sospesi[dirigente];
        for i := 1 to s do
            coda[dirigente].signal;
        { poi gli impiegati }
        s := sospesi[impiegato];
        for i := 1 to s do
            coda[impiegato].signal;
end

begin { inizializzazione delle variabili }
        sospesi[dirigente] := 0;
        sospesi[impiegato] := 0;
        for i := 1 to C do
        begin
            lettori[i] := 0;
            scrittore[i] := false;
        end
end
```

```
var  s: banca; { il nostro monitor }
     d1, d2, ... : dipendente (dirigente, k, l);
     i1, i2, ... : dipendente (impiegato, j, n);

begin end.
```

Starvation

La soluzione proposta presenta starvation nei confronti dei dipendenti di tipo impiegato, i quali possono essere scavalcati in modo indefinito dai dipendenti di tipo dirigente. Per evitare ciò, si può imporre di alternare la priorità ogni tot di esecuzioni, tenendone conto tramite un contatore.

NOTE

Poiché i dipendenti hanno bisogno di due conti correnti, si è preferito sospenderli tutti in una unica coda, risvegliarli tutti quando un conto corrente si libera, e lasciare che siano essi stessi a ritestare le condizioni in un ciclo while.